BINUS UNIVERSITY

Introduction to Web Application using Node.js

RZ 08-2

Node.js is a driving project of Server-Side JavaScript

Table of Contents

Module #01 : Introduction

Overview: Understanding Server-side JavaScript (SSJS) and Node.js

1.1 Definition

JavaScript is one of popular scripting languages. It is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

Normally, JavaScript is a scripting language that runs at client side. It means that JavaScript is loaded before it runs on client. JavaScript can also used in applications outside Web pages, like PDF documents, site-specific browsers, and desktop widgets.

Server-side JavaScript (SSJS) refers to JavaScript that runs on the server-side. The first implementation of SSJS was Netscape's LiveWire, included in its Enterprise Server 2.0 product, released in 1996. CommonJS is a project to provide common specifications for SSJS development.

SSJS is becoming more popular because of faster Javascript engines and the convenience of using the same language for both client and server. One of popular SSJS driving projects is Node.js.

1.2 Node.js

Node.js is a driving project of SSJS that based on V8 JavaScript engine from Google Chrome. Node's goal is to provide an easy way to build scalable network programs. Node can tell the operating that it should be notified when a new connection is made, and then it goes to sleep. If a new connection occurs, then it will execute the callback. It makes each connection will be a small heap allocation.

1.3 Download, System Requirement, and Installing Node.js

Node.js can be downloaded at its official site http://nodejs.org/. It also contains the manual and documentation of Node.js.

System Requirement and Installing documentation can be accessed in https://github.com/joyent/node/wiki/Installation.

This tutorial uses the v0.5.9 of Node.js windows executable program that can be downloaded at http://nodejs.org/dist/v0.5.9/node.exe.

Module #02 : Building a Simple Node.js Application

Overview: Creating a simple "Hello World" Node.js application

2.1 Preparing The Server

As noted above, this tutorial uses the v0.5.9 of Node.js windows executable program. You can download it at http://nodejs.org/dist/v0.5.9/node.exe. Save the node.exe file to a directory in your drive. The program can be run using Windows Command Prompt.

2.2 HelloWorld.js

1. Create a HelloWorld.js file in the same directory with the node.exe file.

2. Open the HelloWorld.js file and add the following code listed below.

```
var http = require("http");

http.createServer(function(request, response) {
      response.writeHead(200, {"Content-Type": "text/plain"});
      response.end("Hello World\n");
}).listen(8087, "127.0.0.1");
console.log("Server running at http://localhost:8087/");
```

Code `var http = require("http");` is used when we need to use the HTTP server and client. The `require()` method is a simple module loading system in Node. `require()` loads the module by directory based, but it is different for core modules. Core modules are always preferentially loaded if their identifier is passed to `require()`. For the example above, the

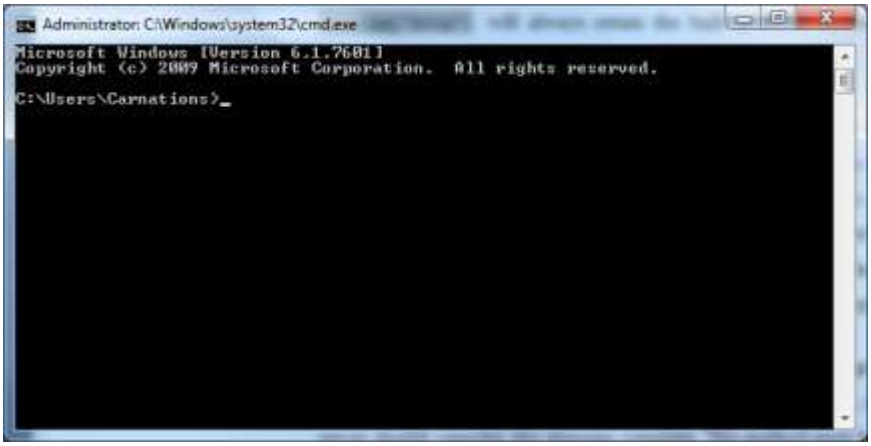
`require("http")` will always return the built in HTTP module, even if there is a file by that name.

We call the `createServer()` method that exists in the HTTP module object. The function creates a server that will handle the request from client. The parameter `function(request, response)` is used to define objects of request and response. In the example above, the `response` object is used to every response to client's request by set the HTTP message header with value `200` as the header type and `"text/plain"` as the content type.

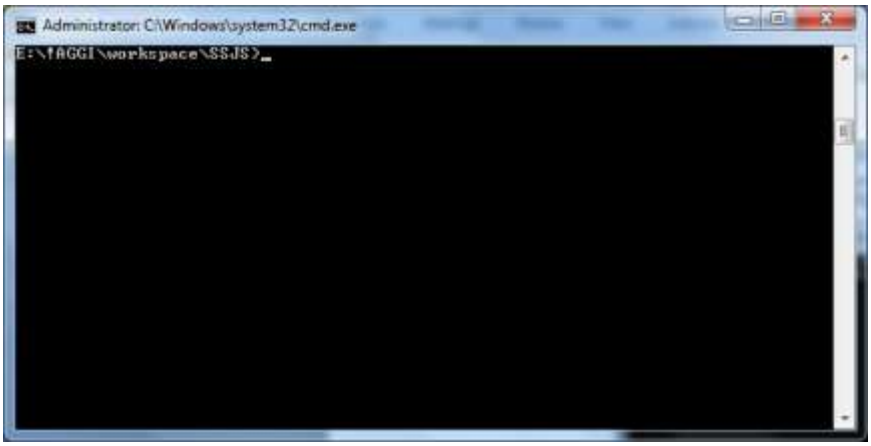
The `response.end([data], [encoding])` is used to send signals to the server that all of the response headers and body has been sent, so that the server should consider this message complete. This method must be called on each response.

`listen(8087, "127.0.0.1")` means that the application will listen to connection to 127.0.0.1 or localhost on 8087 port.

3. Open the Windows command prompt application.



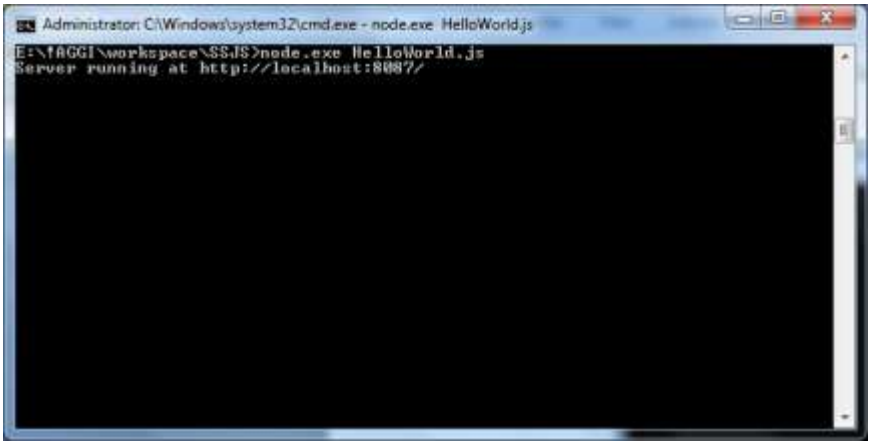
4. Go to directory of the node.exe program.



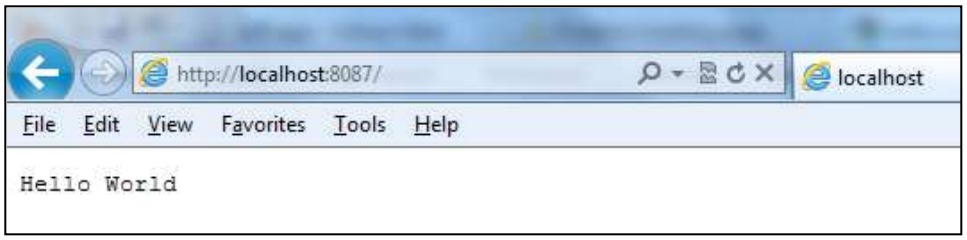
5. Run the script by type and submit the code below in the Windows command prompt.

```
CMD> node.js HelloWorld.js
```

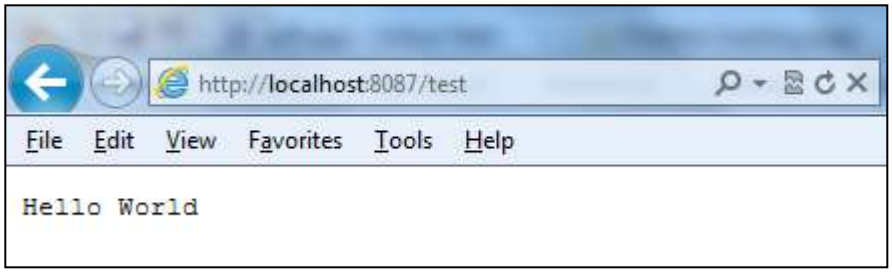
6. "Server running at http://localhost:8087/" text will be printed because of the code `console.log("Server running at http://localhost:8087/");`. That code is usually used to put some text on the console screen.

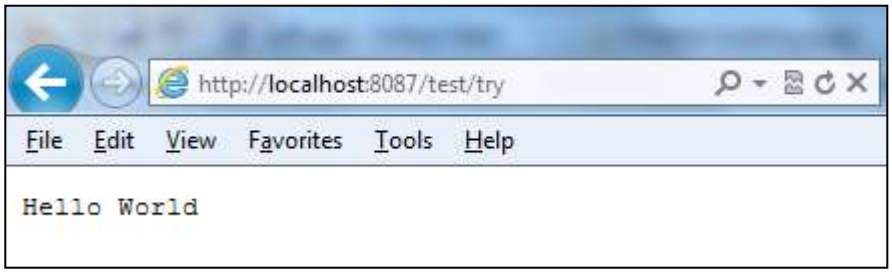


7. Open your browser and access the URL http://localhost:8087/.



8. You will find that HelloWorld.js program always response by returning a "Hello World" plain/text to every request sent to http://localhost:8087/.



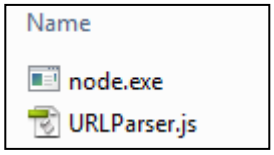


Module #03 : Node.js Application with URL Parameter

Overview: Creating a simple Node.js application that can parse the URL as parameter

3.1 URLParser.js

1. Create an URLParser.js file in the same directory with the node.exe file.



2. Open the URLParser.js file and add the following code listed below.

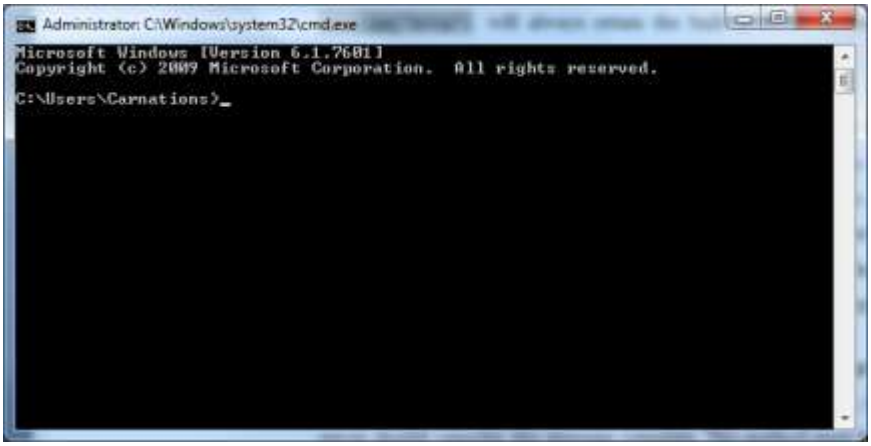
```
var http = require("http"),
      url = require("url");

http.createServer(function(request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});

      //print some details to console
      console.log('Request from: ' +
            request.connection.remoteAddress +
            ' for href: ' + url.parse(request.url).href);

      //parse the URL
      var param = request.url.split('/');
      var message = 'Param count: ' + param.length +
            '\nParam: ' + param;

      response.end(message);
```

The code `request.connection.remoteAddress` returns the IP address of the client.

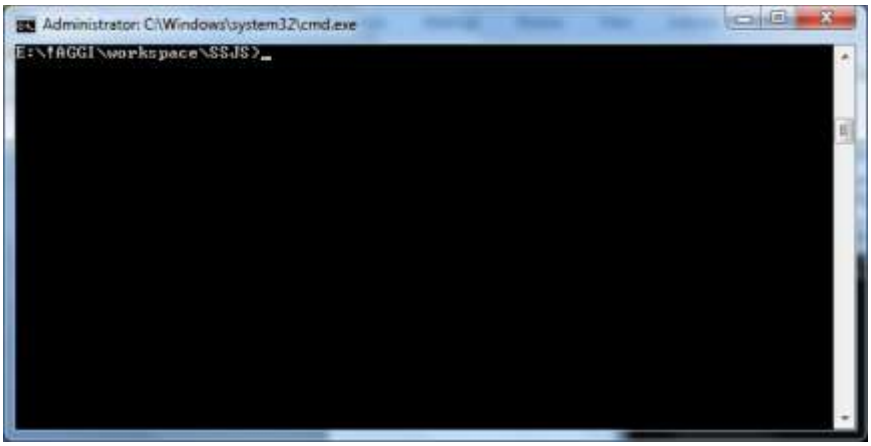
The `url` module has the utilities for URL resolution and parsing. In the example above, the code `url.parse(request.url)` is used to return a URL object from the inputted URL string. The URL object has some fields depend on the URL string. See the documentation at http://nodejs.org/docs/v0.5.6/api/url.html for further detail.

Node accepts every HTTP request with the `listen()` function. So that we must define what must to do with the URL sent by client. By parsing the URL string, it means that we can get every word separated by "/" character.

3. Open the Windows command prompt application.



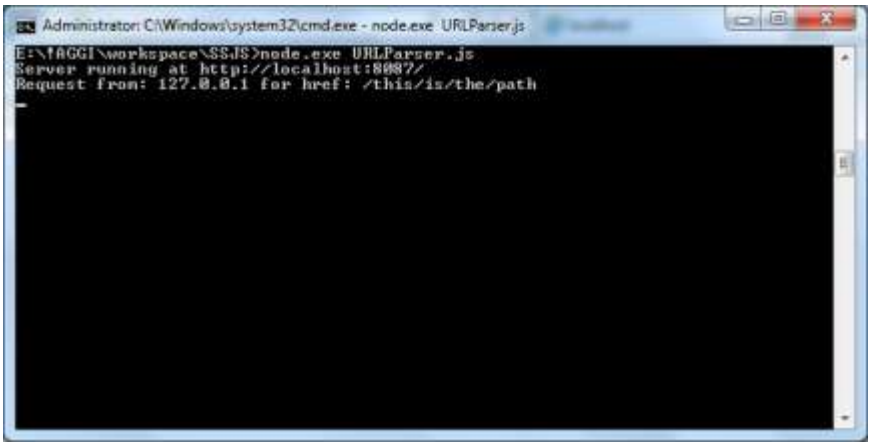
4. Go to directory of the node.exe program.



5. Run the script by type and submit the code below in the Windows command prompt.

```
CMD> node.js URLParser.js
```

6. If a client connected to the Node, some detail about the client will be printed on the console. The picture below show the example.



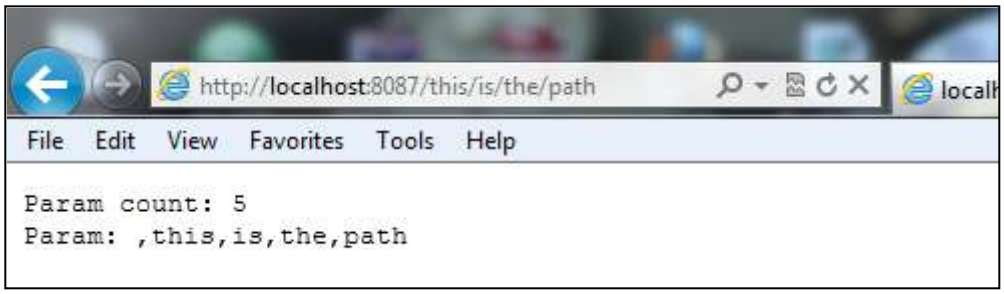
"Request from: 127.0.0.1 for href: /this/is/the/path" is printed on the console, that means a client with 127.0.0.1 IP address requested a http://[host]/this/is/the/path URL.

7. We have made a code that split the URL string by "/" character using this code.

```
//parse the URL
var param = request.url.split('/');
var message = 'Param count: ' + param.length +
      '\nParam: ' + param;

response.end(message);
```

The result of splitting the URL string will be an array of string. In the example, we print the array length and the entire array.

8. If we look at our browser, the response of the node server will be like this.



The example shows that if we request the http://localhost:8087/this/is/the/path URL will be returned as an array of five strings. The first array will always be empty because the URL string in `request.url` will be like this.

```
/this/is/the/path
```

It means that there is an empty string before the first "/" character.

Module #04 : Hosting HTML Pages using Node.js

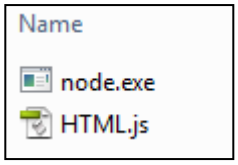
Overview: Understanding the way Node.js returning an HTML page based on the URL given.

4.1 HTML Pages in Node

Node will reponse the same for each URL request that sent by clients. In order to make an Node application that can reponse any requests given with HTML pages based on the URL given, the application must have the ability to browse the HTML page requested and return it as a response.

4.2 HTML.js

1. Create an HTML.js file in the same directory with the node.exe file.



2. Open the HTML.js file and add the following code listed below.

```
var http = require('http'),
      url = require('url'),
      sys = require('util');

var handler = require('./lib/Handler.js');

http.createServer(function(request, response) {

      try {
            console.log('Request from: ' +
                  request.connection.remoteAddress +
                  ' | href: ' +
url.parse(request.url).href);

            handler.handle(request, response);

      } catch (e) {
            sys.puts(e);
            response.writeHead(500);
            response.end('Internal Server Error');
      }

}).listen(8087, "127.0.0.1", function () {
```

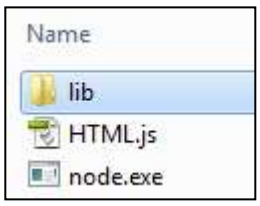
See that a `try { } catch (e) { }` block code has been added. It will handle any exception happens in the `try { }` block by printing the exception to console and returning a 500-HTTP-response with "Internal Server Error" message.

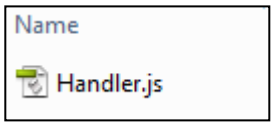
The `util` module has some utilities to help you code. One of its great function is the `util.format()` function. It returns a formatted string using the first argument as a `printf`-like format. In the example above, the `util` module used to call `puts()` method to print something to console instead of using `console.log()` method. See the documentation at http://nodejs.org/docs/latest/api/util.html for further detail.

As mentioned before, Node accepts every HTTP request with the `listen()` function. So that we must define what must to do with the URL sent by client by parsing the URL like we did in the previous module.

4.3 Handler.js

1. Create a javascript file in lib directory and rename it to Handler.js.





2. Open the Handler.js file and add the following code listed below.

```
var fs = require('fs'),
      path = require('path');

this.handle = function (request, response) {
      var renderHtml = function (content) {
            response.writeHead(200, { 'Content-Type':
'text/html' });
            response.end(content, 'utf-8');
      };

      var notFound = function () {
            response.writeHead(404, { 'Content-Type':
'text/plain' });
            response.end("404 Not Found\n");
      };

      var parts = request.url.split('/');
      var page_rest = parts.slice(1).join('/');

      if(request.url == '/') {
            fs.readFile('./web/index.html', function (error,
content) {
                  console.log('opening homepage');
                  if(error) {
                        notFound();
                        return;
                  }
                  else {
                        renderHtml(content);
```

```
      else {
            console.log('opening ' + page_rest);
            fs.readFile('./web/' + page_rest, function
(error, content) {
                  if(error) {
                        notFound();
                        return;
                  }
                  else {
                        renderHtml(content);
                  }
            });
      }
```

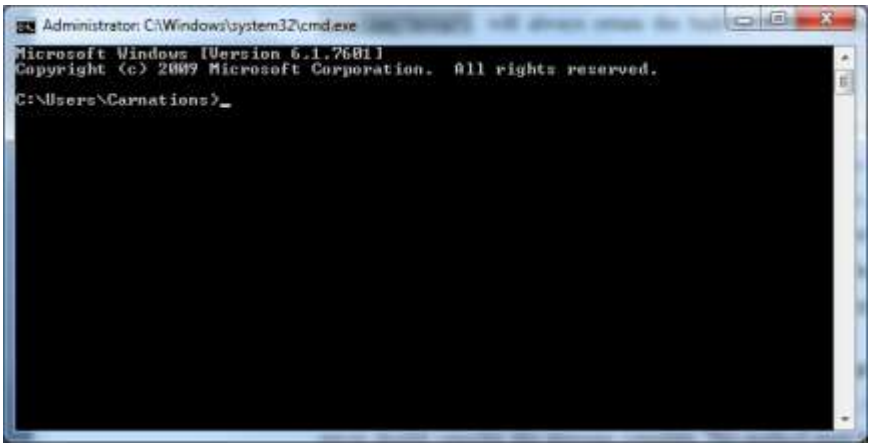
Handler.js will handle any request to the server based on the URL given. There are two main functions that will be the response to client.

```
      var renderHtml = function (content) {
            response.writeHead(200, { 'Content-Type':
'text/html' });
            response.end(content, 'utf-8');
      };

      var notFound = function () {
            response.writeHead(404, { 'Content-Type':
'text/plain' });
            response.end("404 Not Found\n");
```

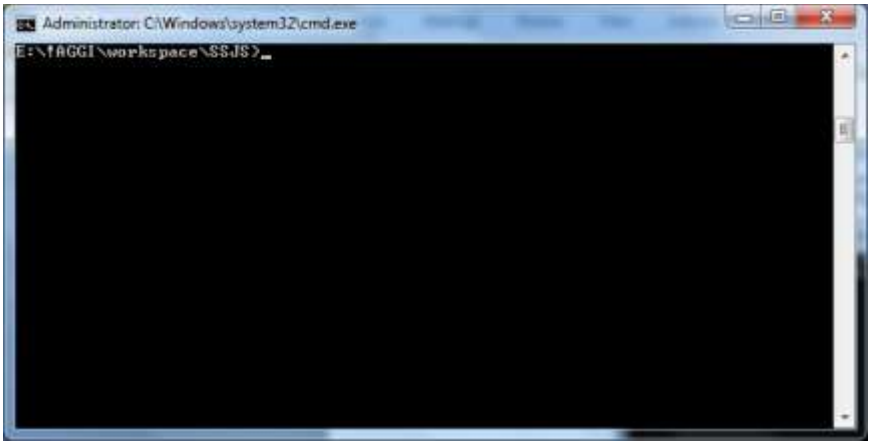
The `renderHtml(content)` function will send a response as a 'text/html' with the content given as the parameter and the `notFound()` function will send a response of 404-HTTP-Not found.

4.4 Run the Application

1. Open the Windows command prompt application.



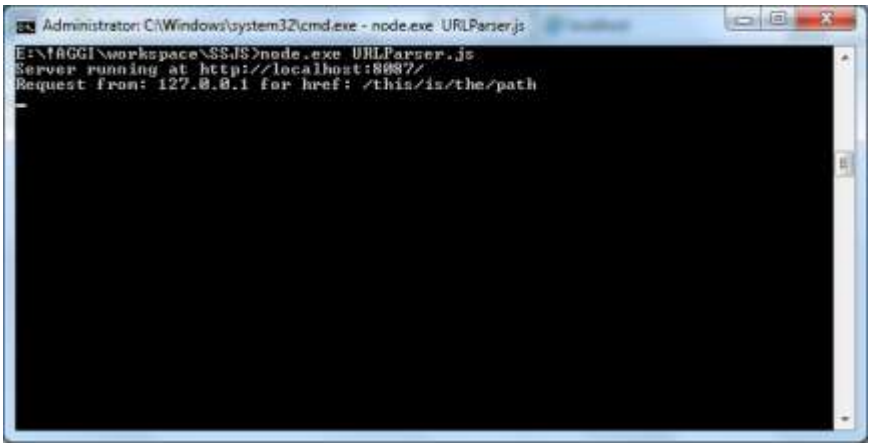
2. Go to directory of the node.exe program.



3. Run the script by type and submit the code below in the Windows command prompt.

```
CMD> node.js HTML.js
```

4. If a client connected to the Node, some detail about the client will be printed on the console. The picture below show the example.



5. Look at this block of code in Handler.js.

```
	var parts = request.url.split('/');
	var page_rest = parts.slice(1).join('/');
```

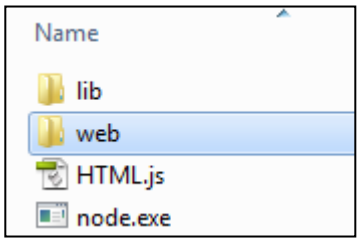
The URL given will be divided into array based on "/" character and restored into `parts` variable. The elements from #1 to #end will be restored as a string into `page_rest` variable.

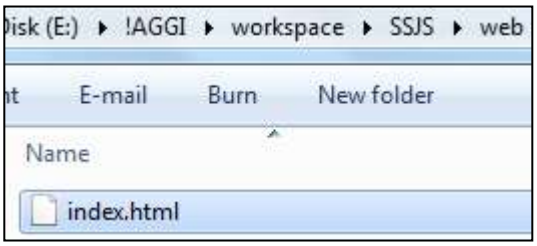
6. Look at this block of code in Handler.js.

```
	if(request.url == '/') {
		fs.readFile('./web/index.html', function (error,
content) {
			console.log('opening homepage');
			if(error) {
				notFound();
				return;
			}
			else {
				renderHtml(content);
			}
		});
```

If the client URL contains the hostname only (like http://localhost:8087/), this application will read a file named "index.html" in [root]/web directory and send a response of a HTML page that contains "index.html" content. So

we need to prepare an "index.html" file inside "web" directory. This page will rule as the homepage.

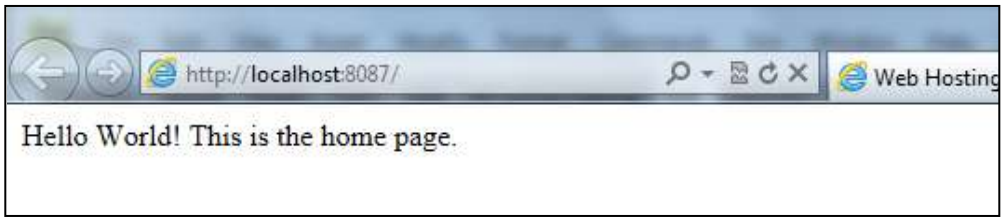




Inside "index.html" file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Web Hosting using Node.js</title>
</head>

<body>
      <div>Hello World! This is the home page.</div>
```

This is what will happen if there is a client request of the URL http://localhost:8088/ using a browser.

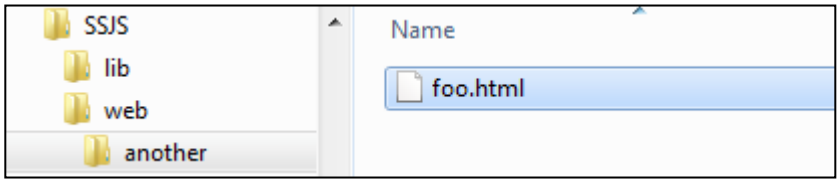


7. Look at this block of code in Handler.js.

```
      else {
            console.log('opening ' + page_rest);
            fs.readFile('./web/' + page_rest, function
(error, content) {
                  if(error) {
                        notFound();
                        return;
                  }
                  else {
                        renderHtml(content);
                  }
            });
```

If the client URL contains the hostname followed by the HTML file URL, this application will search the file URL in 'web' directory. If the file is found, it will be returned as an HTML-response, otherwise the `notFound()` will be called. It means if we want to use this application as a server of HTML pages, the root directory will be the "web" directory.

Example:



Module #05 : Hosting Static Files using Node.js

Overview: Understanding the way Node.js returning a binary file instead of HTML page.

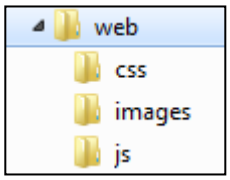
5.1 Static File in Node

If you have tried using external javascript or css file in module #04, you must have found that the external files can't be loaded. It happens because HTTP response of the application in the previous module only send 'text/html' content-type or 400-HTTP-Not found response.

In this module, we will add some code in order to make the application made before can also send responses with binary file.

5.2 External Files

1. Create three new folders under "web" directory like listed below.



2. Create a css file under "css" directory and create a javascript file under "js" directory. You can also copy an image into "images" directory. Put some code that will be used by "index.html". The examples are listed below.

/web/css/style.css

```
@charset "utf-8";

body {
	font-family: Tahoma, Arial;
	font-size: 12px;
	line-height: 1.2;
	color: #000;
	background-color: #FFC;
}

.myStyle {
	font-color: #900;
	font-weight: bold;
	font-size: 14px;
```

/web/images/btn.png

/web/js/script.js

```
function clickMe() {
	alert('You clicked the button !!');
}
```

/web/index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />

<title>Web Hosting using Node.js</title>

<link href="css/style.css" rel="stylesheet" type="text/css"
/>

<script type="text/javascript" src="js/script.js"></script>
```

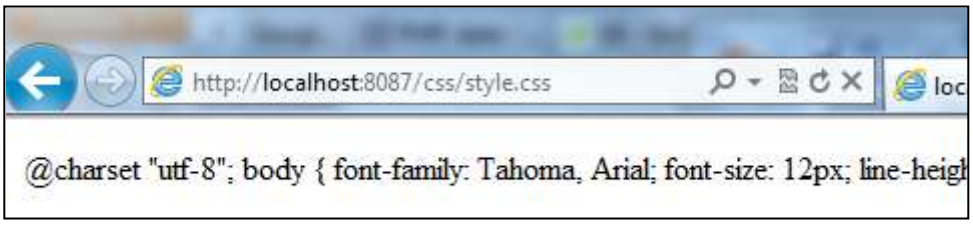
```
<body>

      <div class="myStyle">Hello World! This is the home
page.</div>

    <input type="image" src="images/btn.png"
onclick="clickMe()" value="Click Me" />

</body>
```

3. This is what will happen if there is a client request of the URL http://localhost:8088/ using a browser.

It shows that the image can be loaded. Try to click the button.

External javascript is loaded as well. But realize that the external stylesheet isn't loaded. Every file is loaded as HTML.

4. This is what will happen if we open the stylesheet and javascript file directly from a browser.



5.3 Add Some Code to Handler.js

1. Open the Handler.js file and add some new code.

```
var fs = require('fs'),
      path = require('path');

this.handle = function (request, response) {
      var renderHtml = function (content) {
            response.writeHead(200, { 'Content-Type':
'text/html' });
            response.end(content, 'utf-8');
      };

      var notFound = function () {
            response.writeHead(404, { 'Content-Type':
'text/plain' });
            response.end("404 Not Found\n");
      };

      var parts = request.url.split('/');
      var rest = parts.slice(2).join('/');
```

```
	if(request.url == '/') {
		fs.readFile('./web/index.html', function (error,
content) {
			if(error) {
				notFound();
				return;
			}
			else {
				renderHtml(content);
			}
		});
	}
	else if(parts[1] != 'css' && parts[1] != 'images' &&
parts[1] != 'js') {
		console.log(page_rest);
		fs.readFile('./web/' + page_rest, function
(error, content) {
			if(error) {
				notFound();
				return;
			}
			else {
				renderHtml(content);
			}
		});
	}
	else {
		path.exists('./web/' + parts[1] + '/' + rest,
function(exists) {
			if(!exists) {
				notFound();
				return;
			}

			fs.readFile('./web/' + parts[1] + '/' +
rest, 'binary', function(error, content) {
				response.writeHead(200);
```

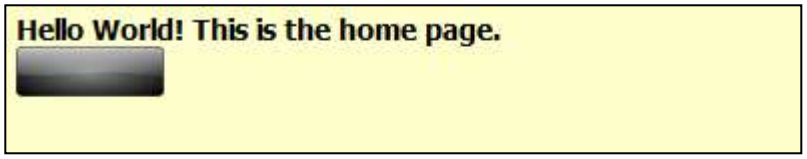
Note:

Added text

Changed text

Now this application will response a HTML file for every URL request, *except* for URL requests began with 'css', 'images', or 'js' like http://localhost:8087/css/style.css, http://localhost:8087/images/logo.gif, or http://localhost:8087/js/script.js.

2. Now, try to open the homepage again.



Module #06 : Communicate using AJAX

Overview: Creating a simple Node.js application that communicate through AJAX.

6.1 AJAX

Asynchronous JavaScript and XML has been very popular in web development. It allows a client to send a request to the server without loading any pages. The response from server will not make the client refreshing any pages also. Using AJAX in developing a Server-side JavaScript is claimed to be faster than any other languages written in server. It is because the client and the server will communicate with the same language. So the translation phase is bypassed.

6.2 HTML Page that Send Request using AJAX

1. This module will use the same project that created in the previous module. Open the "HTML.js" file and replace it with this following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>AJAX using Node.js</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquer
y.min.js"></script>
<script type="text/javascript">
function sendAjax() {
      $.ajax({
            type: "POST",
            url: 'action/' + $('#toDo').val(),  // GET
            contentType: 'text/json',
            dataType: "json",
            // POST
```

```
            success : function(result) {
                  if(result.success) alert("SUCCESS : " +
result.message);
                  else alert("FAILED : " + result.message);
            },
            error: function(jqXHR, textStatus, errorThrown)
{ alert("Internal Server Error"); }
      });
}
</script>
</head>

<body>
      Do this : <input type="text" id="toDo" />
    <input type="button" onclick="sendAjax()" value="Send"
/>
```

I use jQuery AJAX to make it easier to use AJAX (especially because of the cross-browser ability). Instead of using jQuery file hosted online, you can download the latest jQuery here, and replace

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.
min.js"></script>
```

with this following.

```
<script type="text/javascript" src="js/jquery-
1.5.1.min.js"></script>
```

2. Look at this block code.

```
<script type="text/javascript">
function sendAjax() {
	$.ajax({
		type: "POST",
		url: 'action/' + $('#toDo').val(),  // GET
		contentType: 'text/json',
		dataType: "json",
		// POST
		data: JSON.stringify({ Param1: "this is first
param", Param2: 1234, Param3: true }),
		success : function(result) {
			if(result.success) alert("SUCCESS : " +
result.message);
			else alert("FAILED : " + result.message);
		},
		error: function(jqXHR, textStatus, errorThrown)
{ alert("Internal Server Error"); }
	});
```

This application will send request to http://locahost:8087/action/ followed by word typed in the textbox. If it is needed to send POST parameters, we can put them in `data:` attribute.

The success function will handle the response as soon as it is received from server.

6.3 Handling AJAX Requests

1. Open "Handler.js" file and update it with this following code.

```
var fs = require('fs'),
	path = require('path');

this.handle = function (request, response) {
	var renderHtml = function (content) {
		response.writeHead(200, { 'Content-Type':
'text/html' });
		response.end(content, 'utf-8');
```

```
	var notFound = function () {
		response.writeHead(404, { 'Content-Type':
'text/plain' });
		response.end("404 Not Found\n");
	};

	// function that return JSON object
	var respJSON = function (response, object) {
		response.writeHead(200);
		response.end(JSON.stringify(object));
	};

	var parts = request.url.split('/');
	var rest = parts.slice(2).join('/');
	var page_rest = parts.slice(1).join('/');

	if(request.url == '/') {
		fs.readFile('./web/index.html', function (error,
content) {
			if(error) {
				notFound();
				return;
			}
			else {
				renderHtml(content);
			}
		});
	}
	// check if the requested URL started with 'action',
it means that
	// the request is an ajax request
	else if(parts[1] == 'action') {
		// check whether the request method is a POST
method
		if(request.method === 'POST') {
			// get the data in request body (if using
POST)
			var temp = '';
```

```
                request.on('end', function() {
                    var data = {};
                    var action = parts[2];

                    data = JSON.parse(temp);
                    console.log('action to \'' + action
+ '\', data: ' + JSON.stringify(data));

                    var resp = {};
                    // if you need to check what word
follows the "/action" URL
                    // if the action is work
                    if(action == 'work') {
                        // initialize 'resp' variable
that will be used to be the response
                        resp = { success: true,
message: 'You send Work' };
                    }
                    else {
                        resp = { success: false,
message: 'You send ' + action };
                    }
                    // you can add other additional
actions here

                    // return the resp variable as
response
                    respJSON(response, resp);
                });
            }
            else {
                notFound();
            }
        }
      else if(parts[1] != 'css' && parts[1] != 'images' &&
parts[1] != 'js') {
            console.log(page_rest);
            fs.readFile('./web/' + page_rest, function
(error, content) {
                if(error) {
                    notFound();
                    return;
                }
```

```
      else {
            path.exists('./web/' + parts[1] + '/' + rest,
function(exists) {
                  if(!exists) {
                        notFound();
                        return;
                  }

                  fs.readFile('./web/' + parts[1] + '/' +
rest, 'binary', function(error, content) {
                        response.writeHead(200);
                        response.end(content, 'binary');
                  });
            });
      }
```

Note:

`Added text`

`Changed text`

The `respJSON()` function will convert any object given as parameter to a JSON string and send it to client as the response.

The body of the message sent from client need to be checked if the client send request using POST method.

2. Here is the example of the result.